Secure Processing

By Jason Grembi

Secure Processing

- Objectives
  - Secure Error Handling
  - Secure Deployment
  - Secure Storage

Secure Processing

- Secure Error Handling
  - Motivated attackers like to see error messages and look for information that may give clues
    - Application errors expose a lot of information about the code and its environment
    - Errors can tell attackers database information such as table names and columns, names of objects used, code location, Web server information, and even IP addresses

Secure Processing

- Two Types of Errors
  - **Compile-time** errors are found at compile time and the code will not execute until they are fixed
  - Run-time errors occur when a data flow does not flow as expected

Secure Processing

- Handling Errors
  - The following are the steps that need to be performed while handling errors:
    - Code your own routine to catch errors
    - Create application-specific exceptions
    - Manage the views

Secure Processing

- Code Your Own Routine
  - Control the logic by writing an error-handling routine that catches specific run-time errors
  - In an error-handling routine, every code statement has to be surrounded by traditional try{} catch{} blocks so that the appropriate exception can be caught and handled
  - There is no such thing as error-free code; all languages throw errors and all applications have errors – catch them before they catch you.

```
/**
   * This method is called to add a refer a friend contact into the a database
   * @return void
   * @param myForm (ReferFriendForm) An input object
   * @exception Exception An SQL exception occurred selecting against table.
   */
  public static void getReferFriend(ReferFriendForm myForm)
              throws Exception {

      final String METHOD = "getAddReferFriend()";
      debug(CLASS + "." + METHOD + "==>Begin");

      if (myForm.getClass()!= ActionForm){
          return;
      }

          UserBean userBean = myForm.getUserBean();
          ApplicationLogger logger = ApplicationLogger.getInstance();

      String query = "SELECT  * FROM SOME_TABLE WHERE I_USER_FNAME " +
                  " '"+userBean.getFirstName()+"'";

      try {

      debug("SQL looks like this \n"+query);

      } catch (Exception e) {

          debug(CLASS + "." + METHOD + "==>SQLException occurred preparing SQL statement:" +
              query);

              logger.writeMsgLog("\nerror occured"+CLASS + "." + METHOD +e.getMessage());

              ApplicationError ae = new ApplicationError();
              ae.setException(e);
              ae.setUserViewMessage("Unexpected Error Occured");
              ae.setErrorMessage(e.getMessage());
              throw ae;

      } finally {
          debug("Done with "+METHOD +CLASS);
          }
  }
```

Secure Processing by Jason Grembi

Secure Processing

- Creating Application-Specific Exceptions
  - When catching exceptions, the application needs to determine what to do when the exception occurs
  - To help make this decision, an encapsulated error class can be created to handle events for all use cases/misuse cases within the application
  - After the ApplicationError class is created, it can be embedded within the code in the catch blocks
  - Exception handling is the cornerstone for all secure code

```java
public class ApplicationError extends Exception{

    /** Error Message */
    private String errorMessage;

    /** Exception */
    private Exception exception;

    /** Class Name in which exception occurred. */
    private String className;

    /** Method Name in which exception occurred. */
    private String methodName;

    /** User Friendly Error Message. */
    private String userViewMessage;

    /**
     * A default constructor.
     */
    public ApplicationError() {
        super();
    }
    /**
     * A constructor that a String object that contains
     * an error message.
     */
    public ApplicationError(String errorMessage) {
        super(errorMessage);
    }

    /**
     * Returns the className.
     * @return String
     */
    public String getClassName() {
        return className;
    }

    /**
     * Returns the errorMessage.
     * @return String
     */
    public String getErrorMessage() {
        return errorMessage;
    }

    /**
     * Returns the exception.
     * @return Exception
     */
    public Exception getException() {
        return exception;
    }
```

Secure Processing

- Managing the Views
  - What you choose to tell the user in error messages is up to you, but it should specifically benefit them
  - The developers need to see the raw error codes, the invalid SQL statements, or the object names of the defective code
  - The users, on the other hand, should see only friendly messages that gracefully tell them only what they need to know

Secure Processing

```
/**
 *
 * @author Jason
 */
public class ApplicaitonErrorHandler {

    /**
     * Constructor for SC.
     */
    public ApplicaitonErrorHandler() {
        super();
    }

    public ActionForward execute(Exception ex,ExceptionConfig ae,ActionMapping mapping,
                      ActionForm formInstance,HttpServletRequest request,HttpServletResponse response)
                     throws ServletException {
     ApplicationLogger logger = ApplicationLogger.getInstance();
     logger.writeLog(ex,request);

      request.setAttribute("myException",ex);
      return new ActionForward(ae.getPath());
   }

}
```

Secure Processing

- Creating Self-Monitoring Code
  - You want to be able to trace every step, action, and error the code executes.
  - Examples of paths of execution, events include the number of times a single user has tried to log on, submit values, or request sensitive information that exceeds “normal” conditions

```
                }
                Location gloc3=gridApi.readByLocationID(gridCompanyId);

             if (gloc.getUniqueIdName().equals(company.getUniqueIdName()))
                 matchCompanies.add(new CompBean(gloc, gloc3.getUltimateUniqueId()));
             }
         }
    } else if (ALIAS_ULT_LOC_CODE != null && ALIAS_ULT_LOC_CODE.trim().length() > 0){
        logger.debug(CLASSNAME+" call ==>readUltimate");
        request.setAttribute("aliasRecs", "true");
        Location gloc=gridApi.readUltimate(ALIAS_ULT_LOC_CODE, company.getUniqueIdName());
    if (gloc != null && gloc.getUltimateUniqueId().equals(gloc.getUniqueId())){
            gridCompanyId = gloc.getLocationId();
            if(gloc.getAuthorityId() != -1)
                    gridCompanyId=gloc.getAuthorityId();
            if (gloc.getUltimateUniqueId() != null) {
                Location gloc2=gridApi.readByUniqueID(gloc.getUltimateUniqueId(), gloc.getUniqueIdName());
                if(gloc2 != null && gloc2.getAuthorityId() != -1)
                        gridCompanyId=gloc2.getAuthorityId();
            }
            Location gloc3=gridApi.readByLocationID(gridCompanyId);
    matchCompanies.add(new CompBean(gloc, gloc3.getUltimateUniqueId()));
    }
    } else if (ALIAS_LOC_CODE != null && ALIAS_LOC_CODE.trim().length() > 0){
        logger.debug(CLASSNAME+" call ==>readByUniqueID");
        request.setAttribute("aliasRecs", "true");
        Location gloc=gridApi.readByUniqueID(ALIAS_LOC_CODE, company.getUniqueIdName());
if (gloc != null){
    logger.debug(CLASSNAME+" gloc != null");
            gridCompanyId = gloc.getLocationId();
            if(gloc.getAuthorityId() != -1)
                    gridCompanyId=gloc.getAuthorityId();
            if (gloc.getUltimateUniqueId() != null) {
                logger.debug(CLASSNAME+" call ==>getUltimateUniqueId");
                Location gloc2=gridApi.readByUniqueID(gloc.getUltimateUniqueId(), gloc.getUniqueIdName());
                if(gloc2 != null && gloc2.getAuthorityId() != -1)
                        gridCompanyId=gloc2.getAuthorityId();
            }
            logger.debug(CLASSNAME+" call ==>readByLocationID");
            Location gloc3=gridApi.readByLocationID(gridCompanyId);
            if (gloc3.getLegalName() != null && gloc3.getLegalName().trim().length() > 0) {
```

Secure Processing

- Secure Deployment
  - Many manual tasks
  - To many things to go wrong
  - Usually done in late hours of over weekend

Secure Processing

- Secure Deployment
  - Many manual tasks
  - To many things to go wrong
  - Usually done in late hours of over weekend

Secure Processing

- Build Management
  - **ANT** (Another NeatTool), located at *http://ant.apache.org/*, is an XML-based tool that calls out targets (or specific tasks) in a treelike structure
  - Developers use ANT version control tools when doing code deployments from one environment to another
  - ANT replaces all the manual tasks that developers do before deploying code
    - *FTP (FileTransfer Protocol) code*: Automatically FTPs code files from one machine to another
    - *Get code*: Automatically interfaces with version control software
    - *Move code*: Takes files or directories and moves them anywhere desirable
    - *Message code*: Changes parameters automatically
    - *Compile code*: Turns ASCII files into class files (binary)

```
<project name="appName" default="jarApp" basedir="C:\ssd\ant">


<!--  create directoy structure -->
  <property file="C:\ssd\ant\${propfile}"/>
  <property name="appName" value="c:" />
  <property name="project" value="\ssd\ant" />
  <property name="build" value="\build" />
  <property name="deployDir" value="c:\ssd\ant\deploy" />
  <property name="src" value="\src" />
  <property name="srcDir" value="${appName}${project}${src}" />
  <property name="buildDir" value="${appName}${project}${build}" />

  <property name="ant" value="C:\ant\apache-ant-1.6.2\lib\ant.jar" />
  <property name="classpath" value="${ant}"/>

<!--  copy all Java Files from src to build and exclude the class files-->
<target name="copyDir" description="copy all Java Files from src to build">
  <copy todir="${buildDir}">
    <fileset dir="${srcDir}" excludes="**/*.class"/>
  </copy>
</target>

<!--  touch all the files -->
  <target name="timestamp" description="adds timestamp to logfile" depends="copyDir" >
    <tstamp>
      <format property="touch.time" pattern="MM/dd/yyyy hh:mm aa"/>
    </tstamp>
    <echo message="refreshScreen script started on ${touch.time}"/>
  </target>

<!--  Turn OFF debugs -->
  <target name="turnOffDebugs" depends="timestamp" if="TURN_OFF_DEBUGS" description="Turn off debugs when Prod Deploy">
    <replace dir="${buildDir}"
     token="private final static boolean DEBUG = true;"
     value="private final static boolean DEBUG = false;"/>
  </target>

<!--  Compile java Files -->
  <target name="javac" depends="turnOffDebugs" description="compiles the java files" >
    <javac srcdir="${buildDir}" destdir="${buildDir}" classpath="${classpath}" debug="on">
    </javac>
  </target>

<!--  Jar up the Files -->
 <target name="jarApp" description="this creates a jar file" depends="javac">
    <jar jarfile="${deployDir}/app.jar" basedir="${buildDir}"/>
  </target>

</project>
```

Secure Processing

- Static Code Analysis
  - Static code analyzers scan (parse) through static code and analyze the code base for security vulnerabilities (such as input fields and buffer overflows)
  - A number of static code analyzers are available on the market
  - Tools come with sophisticated code parsers, but they also have reporting features

Secure Processing

- Code Reviews and Inspections
  - Code reviews are people-intensive verification techniques that are conducted either formally or informally that allow peers to read code statements and look for common security vulnerabilities, such as hard-coded IDs or passwords, and general quality features
  - Reviewing code as a team is actually a great opportunity for the strongest skilled developers to share knowledge, rationale, and guidance with the entry/mid-level developers

Secure Processing

- Code Reviews and Inspections (continued)
  - The following are the steps to a successful code review:
    - The developer of the use case prints out all the code and makes copies for each participant
    - At the meeting, each participant walks through the use case, step-by-step, and the programmer talks about each activity
    - Every code statement is analyzed and reviewed by the participants against a checklist of rules and guidelines
    - As issues or concerns come up during the meeting, the code is highlighted on the hard copy and notes or instructions are taken
    - After the meeting, it is determined if another walk-through is needed or if the business analyst needs to be consulted for any questions

Secure Processing

- Verifying System Documentation
  - Before you put the test case to rest and go on to the next one, verify that all system documentation is finalized and up to date
  - System documentation is a lot like life insurance—you might never need to use it after it is finalized, but the one time you need it to be available and accurate, it will be there for you
  - The following activities will keep the software documentation up to date:
    - Verify that the use case documentation says what the software does
    - Verify traceability on the current use case throughout the whole life cycle
    - Verify the current version
    - Update the Application Guide

Secure Processing

- Introducing Software Upgrades
  - Updating the supporting operating systems, application servers, or tools can be exciting yet scary (in terms of security)
    - Treat any new upgrade as a threat to your architecture
    - New features offer new solutions, and a redesign of your application might be necessary to take advantage of this

Secure Processing

- Introducing New Developers
  - After a few successful years of secure programming on one assignment, developers are likely to go onto another new project and reuse the Application Guide and their skills to help other projects get started
  - The introduction of new people can be a security risk during software maintenance
    - They don’t have the background knowledge that the developers who wrote the software do

Secure Processing

- Proving Your Work
  - Your proof will be evidenced in the following:
    - *Application Guide*
    - *Use case*
    - *Misuse case*
    - *Code reviews*
    - *Test scripts*
    - *Deployment*
    - *Secure Logs*

Secure Processing

- Secure Storage
  - A code repository is used for code storage
  - Code repositories allow that programmer to check in the code from the workstation and into a centralized data house

Secure Processing

- Secure Storage
  - Offers an opportunity to centralize the backups of source code and ensure that current backups of the entire repository are available for recovery in case of a failure
    - Code sharing: All other programmers have access to the latest and greatest without stepping on one another's code
    - Versioning/baseline: Developers can manage which features go out in the next version
    - Centralizing storage: This allows the code to be deployed from one centralized place

Secure Processing

- Secure Storage
  - Open source code is great to use and highly encouraged within the software development, but don’t assume it is secure
  - Wherever the JAR files (or EXEs and DLLs) are stored, make sure no one else has access rights to that directory

Secure Processing

- Secure Storage
  - Anyone with access to certain directories can un-JAR a file, tamper with a class, then re-JAR that file and put it back in the classpath
  - It would take even very smart developers weeks or months before figuring out a JAR file has been tampered with
  - JAR files almost never get reviewed and most people do not know what *should* be in there anyway

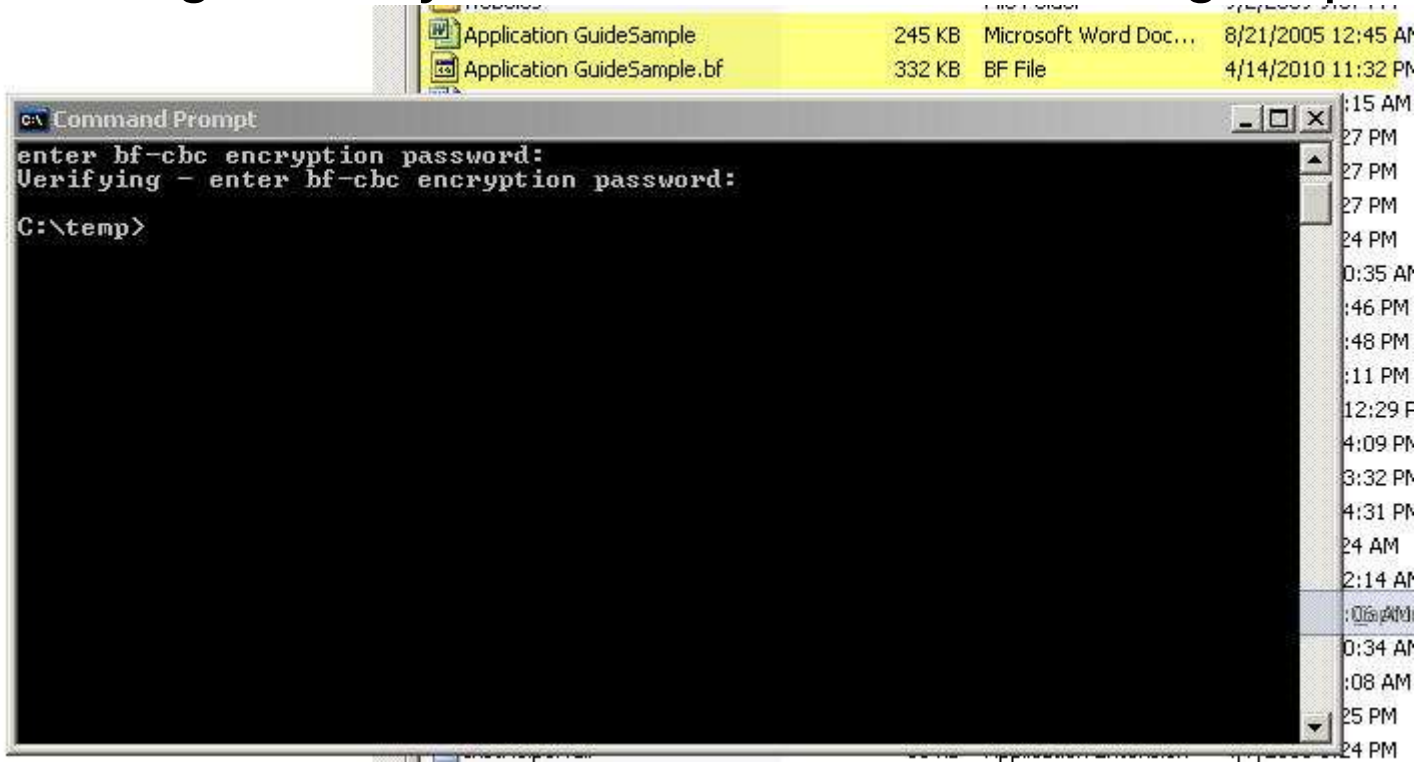
Secure Processing

- Secure Storage
  - A good way to sense if a file has been tampered with is to audit the dates on each file. You can do this by a UNIX Shell job:

```
#!/bin/ksh
 a="2007-10-27"
 b="2006-03-22"
if [[ $((  $(echo $a | tr -d '-')   -  $(echo $b | tr -d '-')  )) -ge 0 ]] ; then
   print "correct"
else
   print "failure"
fi
```

Secure Processing

- Secure Storage
  - A good way to lock down files is through OpenSSL



Secure Processing

- Summary
  - Secure Error Handling
    - Secure Handling
    - Secure Reporting
    - Incident Response
  - Secure Deployment
    - Automated Process
    - Security Scan
  - Secure Storage
    - Locking Down the File Path